Secure Work-From-Home:

Anti-Phishing Browser Extension

Jash Patel

patejash@sheridancollege.ca

Faculty of Applied Sciences and Technology

Sheridan College

1430 Trafalgar Rd, Oakville, ON, Canada

Academic Advisor: Nicholas Johnston and Ali Hassan

nicholas.johnston1@sheridancollege.ca

ali.hassan1@sheridancollege.ca

Abstract:

A remote desktop environment is an alternative solution from on-site work that has gained interest for enterprise desktops due to COVID-19. To ensure, business continuity many organizations have transitioned to a work-from-home approach within a virtual desktop environment. However, based on reports by [1] [2] risks of cyberattacks have drastically increased in remote desktop environments. Although, employee' remote desktop service is secured by the enterprise there are other risk factors that are not protected. This paper conducts detailed research on phishing attack as it is one most optimal attack used by attackers [3] and perform a penetration testing on AWS workspace, a cloud-based virtual desktop platform. Following a successful penetration test, a full analysis is performed on and the vulnerability is secured by creating an Anti-Phishing tool. The research shows the extension applied URL-based and web-content based features for training datasets through the Artificial Neural Networks (ANNS) algorithm. The algorithm then determines if the website is phishing or not and is analyzed how accurate the result is.

Keywords: *Virtual desktop, Neural Networks, Covid-19, URL-based, content-based, extension*

Table of Contents

I. Introduction:

The mid of 2020 COVID-19 pandemic began with lockdowns and challenging many workplaces. In response, many organizations had to transition their staff to remote work in an enterprise-managed remote/virtual desktop environment to maintain business continuity. The transition to cloud services was sudden that resulted in increased cyber-attacks and incidents related to Business Continuity Plan (BCP)[1]. Among the attacks, Remote Desktop Protocol (RDP) and social attacks are the three attacks that caught the attention of many cybersecurity researchers [1][2]. The research conducted by [1] shows phishing attacks tops in breaches [1] by being present in 25% of breaches. After all, a phishing attack is one of the most dangerous social engineered attacks that's been the favourite method for cybercriminals [3]. Therefore, this research project will explore phishing attacks and their characteristics, methodology and behaviours in the virtual desktop environment.

A. What is Phishing?

The word 'phishing' comes from the term *fishing* where the attacker uses a 'bait' and 'fishes' as in waiting for victims to bite exposing their sensitive data [5]. The phishing attack is a type of social engineered attack that involves impersonation to steal sensitive personal information such as credit card data, credentials, identity theft and other sensitive data. It is a fraudulent attempt that takes advantage of human nature in a form of a cyberattack. A kind of mechanism that uses social engineering skills and human nature to destroy trust between the victims and businesses. Phishing attacks have been around for more than 2 decades and despite the effort of the cybersecurity specialists phishing attack still remain the optimal choice of attack for cybercriminals [3][5][6]. Phishing attacks come in a range of different approaches with different purposes and goals. Generally, phishing attacks are used to steal credentials for financial fraud [7]. From 2016 to 2020, 98% of the stolen cards had personal information linked with them. Although, that rating has been declining every year due to the automated approach many organizations using [7]. After all, the Machine Learning approach is a better option than the classic phishing detection approaches; whitelisting and blacklisting because of, scalability and accuracy of the detection [8]. In addition, it is effective against 0-day attacks [8].

B. Phases of Phishing Attack

There are six stages of the phishing attack process. The attack begins by performing reconnaissance to obtain the victim's email. Then a phishing email is sent to the victim with a malicious hyperlink following appropriate phishing technique and the attacker waits for the victim to take a bite. On a successful attack, the data is sent to the attacker to process and analyze useful data.

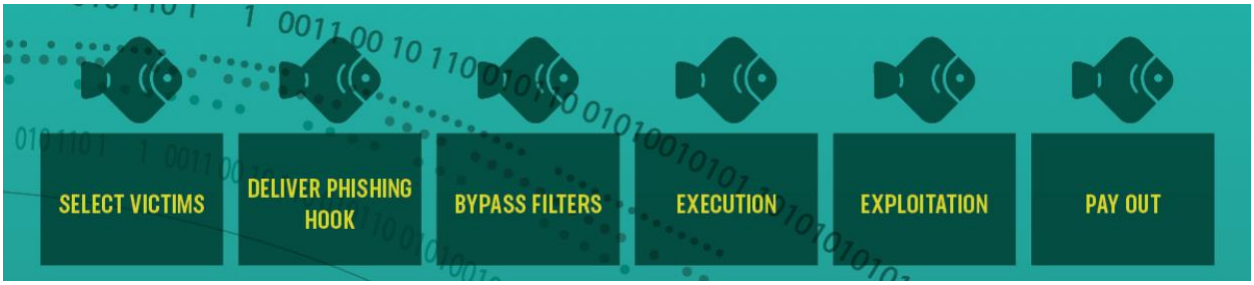

Figure 1: Phases of the phishing attack. Source: [7]

- **Select Victims:** Plan out and perform reconnaissance on a target for an attack.

- Deliver Phishing Hook: Appropriate technique is used and approach is determined. For example, phishing email, vishing, spear-phishing, SMS phishing and many others.
- **Bypass Filters:** Add a phishing link to the E-Mail that will bypass the spam filter when the phishing is sent to the victim.
- **Execution:** The victim clicks on the malicious link and execute the malware or redirect the user to the phishing website.
- **Exploitation:** Phishing websites collect victim's sensitive data including credentials, credit card information, personal information and other sensitive data. But if it is malware, it will collect credentials from the browser/running E-Mail session or execute ransomware or other malware.

C. Phishing Detection approaches

According to the [9], there are two approaches to mitigate phishing attacks:

1. **User training:** Educating and warning users and employees in the enterprise about phishing attacks and its impact on the organization if it is not correctly identified. To correctly identified for possible phishing E-Mails must be reported to the system administrators for blacklisting and whitelisting URLs. This approach may help users distinguish between legitimate and phishing emails but, not an effective solution to prevent or mitigate the phishing attack. After all, human error is bound to occur whether you are tech-savvy users or non-technical users [10].
2. **Software Detection:** the use of the software is an enhanced approach that is better and very reliable compare to user training. With software detection, it is able to analyze multiple technical features before the final decision of blacklisting the website. This method is very beneficial in detecting 0-day attacks and very useful in identifying static and dynamic web-content features which are easily ignored in the user training approach.

D. Software Detection techniques

Rule-based: The rule-based method is used to determine phishing websites by differentiating the URL and web-content features between legitimate and non-legitimate websites. This process gathers various known phishing techniques and characteristics used by the attackers to determine a specific ruleset. In other work, it follows the Intrusion Detection and Prevention framework.

Machine learning: The machine learning approach extracts various features from publicly available datasets as the input for the classifiers. Then, the datasets are used to develop a learning model to be tested on phishing websites. There are many different types of machine learning algorithms used to solve phishing detection problems where the most used ones are decision tree, random forest, SVM, and neural networks.

II. Problem Statement

The transition from on-site work to work-from-home is assumed to be in a secure virtual desktop environment by many organizations. This step was taken to maintain business continuity as per Business Continuity Plan (BCP) [1]. But many organizations neglected to consider security risk factors and their staff became the target for an attack. Although the virtual desktop environment is secured by the enterprise, the involvement of general communications with the organization's network becomes its weakness [1]. In addition, within the secure virtual desktop environment, employees are not limited to

external resources and not under the employer's supervision that can put the virtual desktop environment in an insecure state. In other words, their many risk factors relate to working-from-home and areas which the enterprise may not be able to secure such as social engineered attacks.

Most importantly, a phishing attack, a type of cyber attack that involves human interaction to be executed on the end-user desktop with the intention of a fraudulent attempt. This research will involve penetration testing to exploit phishing vulnerability on the virtual desktop while evading enterprise-like detection. Once the vulnerability is exploited, a secure mechanism will be put in place to prevent a phishing attack.

III. Related Work

There have been many studies and development relate to detecting and preventing phishing attacks due to people always getting tricked by phishing E-Mails [1]. Although there are not many reports or articles related to phishing attacks in a virtual desktop environment, existing studies about phishing prevention should provide me with essential information in developing my solution.

The study conducted by [4], compares eight different classic supervised machine learning algorithms for detecting phishing websites. The eight classifiers used were AdaBoost, CART, Gradient Tree Boosting, k-Nearest Neighbours, Naïve-Bayes, Multilayer Perceptron, Random Forest, and Support-Vector-Machine (SVM). Each of these classifiers has its pros and cons. For these algorithm experiments, they used publicly available phishing websites datasets with the pre-set characteristics of the websites. The researcher used the Scikit Learn library to train and test the classifier with 30 stratified folds [4]. For observation and analysis learning curves were plotted using the cross-validation (CV) function from the Scikit library [4]. The learning curves were then utilized to decide on tuning the hyper-parameters by answering few questions whether "the algorithm learning on training data or memorizing it?" and "is the algorithm prone to overfitting (low bias, high variance) or underfitting (high bias, low variance), or learns 'just right'?"[4]. These questions were asked to determine the accuracy of the algorithms, which later were ranked by accuracy for all datasets used for the experiments. As a result, the experiment concluded with the neural networks outperforming other classifiers with an average accuracy of 0.9742 [4].

The "Phishing-Inspector: Detection & Prevention of Phishing Websites" paper by [17] created a solution similar to what I am looking for. The research talks about the development of a Phishing-Inspector extension that extracts various features from the URL and is compared with phishing and non-phishing websites. The tool checks for the presence of the URL feature in both legitimate and phishing websites datasets. If not found, the system would proceed through ML algorithms whose output is passed on to the SSL checker and domain creation and expiration checker to predict the status of the active website. The result shows among the six classifiers, random forest outperforms with the most accurate result with 93.16%.

Kang Leng Chiew et al.[18] proposed a system that detects phishing websites using the fuzzing technique. The paper leverages the favicon and other URL features to identifying phishing websites. The framework begins by extracting the favicon and appending it to the current URL for a download. Later, the image is compared with analyzed and query to match with image data from the search engine [18]. To accomplish this task custom APU by Schaback is used [18]. For the websites that do not have a favicon, the system extracts other additional URL features such as domain age, dots in URL, Web of Trust

and more. As for the result, Kang Leng Chiew et al.[18] applied their mathematical approach to determine the accuracy of phishing websites.

IV. Solution

The solution to prevent phishing attacks is to develop an anti-phishing browser extension. Although there are many anti-phishing extensions out there, most of them are either outdated, does not fully function, or are not made to function on the virtual desktop. I want to create a browser extension that will not only work on a virtual desktop but will also work on the local desktop. The tool will detect and prevent phishing websites through an automated process by applying a neural networks as the classifier for machine learning on the existing phishing dataset. In addition, the classic URL extraction and Web-content extraction approaches will be implemented alongside as well. The extension will use an open-source plugin to set up communication between client and web servers of the browsing websites and will be supported on Firefox, Chrome and MS Edge platforms.

V. Hypothesis

Based on the literature review, the hypothesis was made that securing phishing attacks would be possible without creating a server-based solution or ever be will be programming in JavaScript, after going through the online articles, related work and reports I had to resort my project by creating an extension. My plan to securing the virtual desktop was using snort, and some monitoring tools. Although, my proof of concept went well as planned.

VI. Risk Assessment:

After completing the research, I noticed that most of the risk factors I mentioned in my proof of concept were on point. For example, time consumption; which was my biggest risk factor for this project. My concern was I would not have enough time to complete my project and worried project tasks might go unplanned.

VII. Technical Details

A. Virtual Desktop Environment Setup

There are two parts to this experimental research. Part I focuses on running penetration testing using open-source social engineering tools against a secure virtual desktop environment to exploit a vulnerability. As for Part II, the phishing vulnerability was secured or mitigated to prevent security issues.

The testing environment I set up was Remnux virtual image running on VMWare Workstation and an AWS Windows 10 Workspace running on cloud associated with virtual private cloud (VPC) and AWS Managed Active Directory for managing users information.

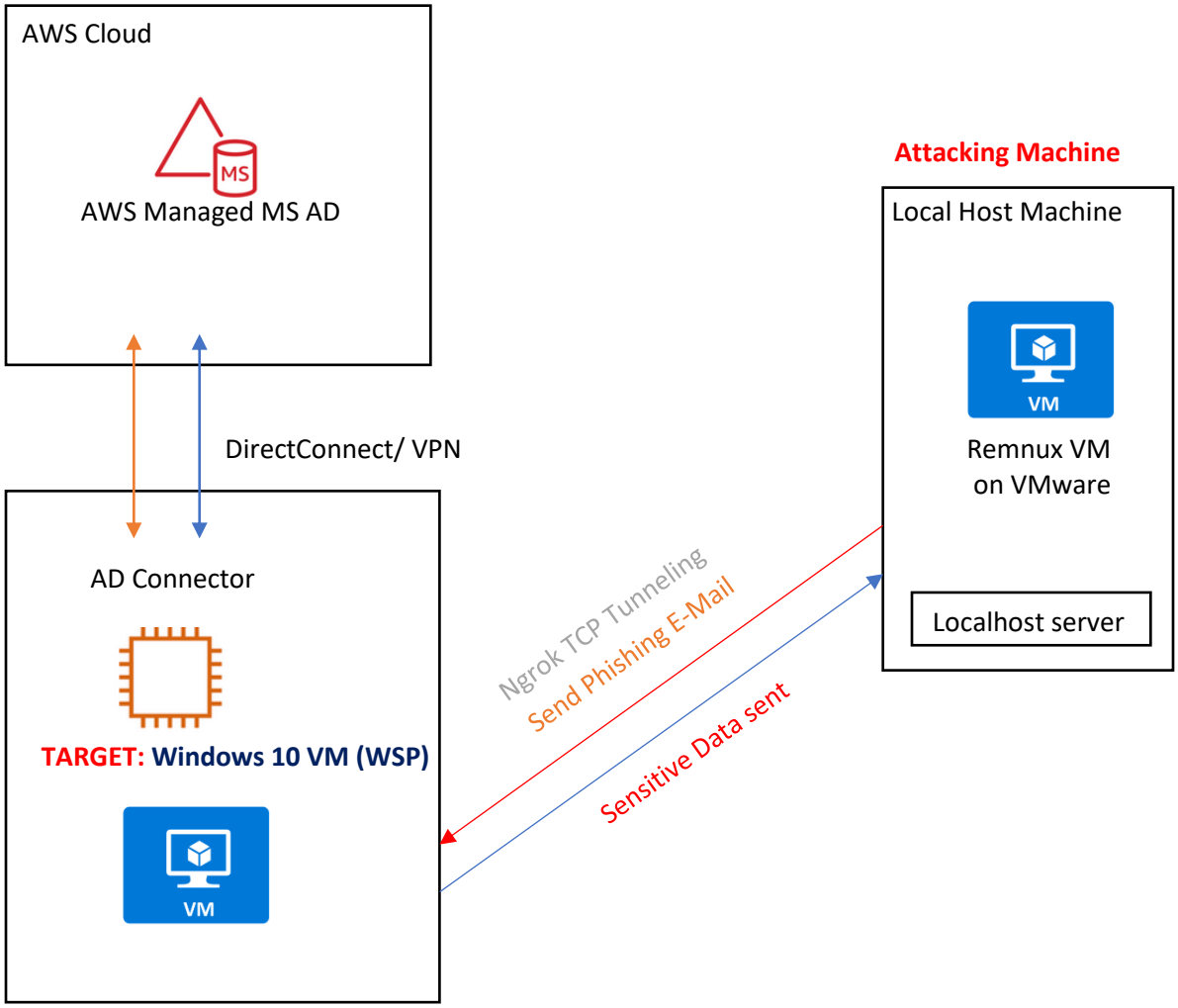

Figure 2: Environment Setup

The Remnux virtual machine was used to create phishing link using Ngrok TCP tunneling and hosting dedicated local server so that the link sent to target would be able to bypass firewall and NAT mapping [11]. Then, the hyperlink was clicked by the user and the link redirected user to phishing site where the user was told to log-in. But, unfortunate for the user, all the sensitive data was sent to localhost. The phishing website was executed on AWS Windows 10 virtual desktop while, the localhost VM was used for logging sensitive data.

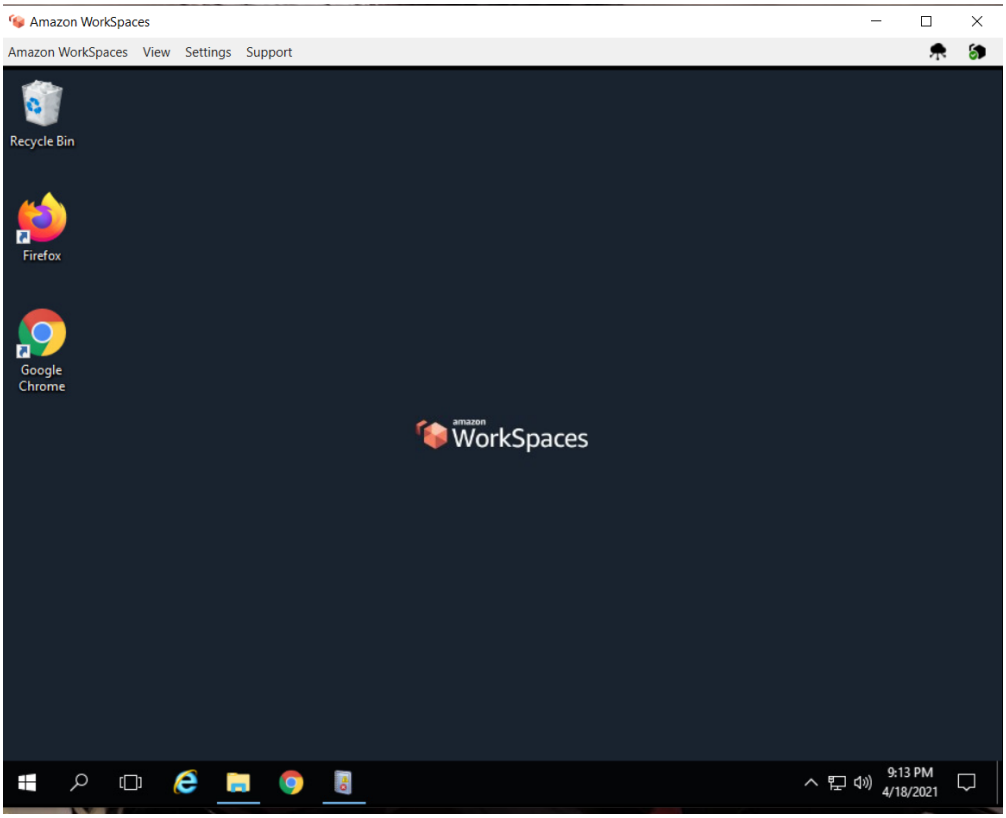

Figure 3: AWS Workspace setup

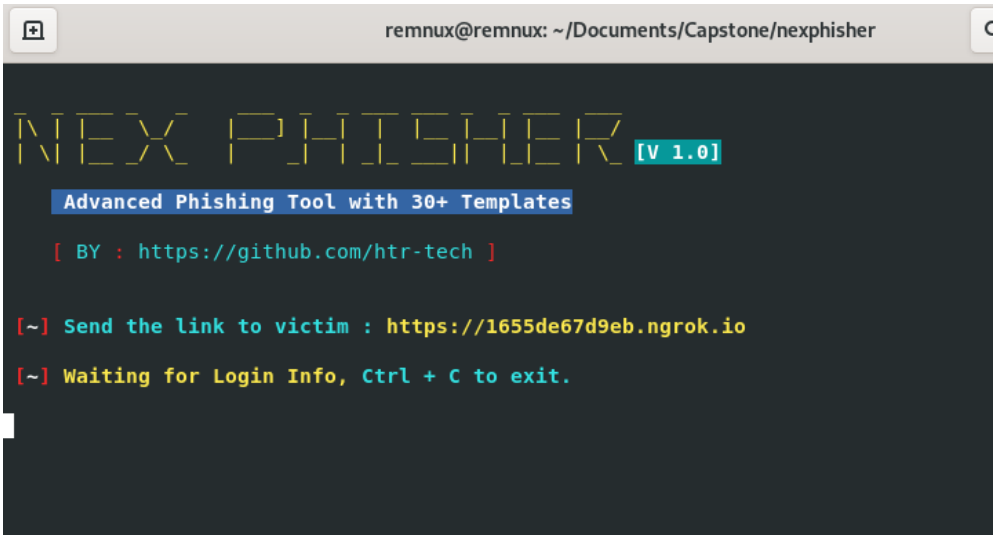

Figure 4: Running Ngrok service for LinkedIn phishing email

.

Figure 5: Select Ngrok for TCP tunneling

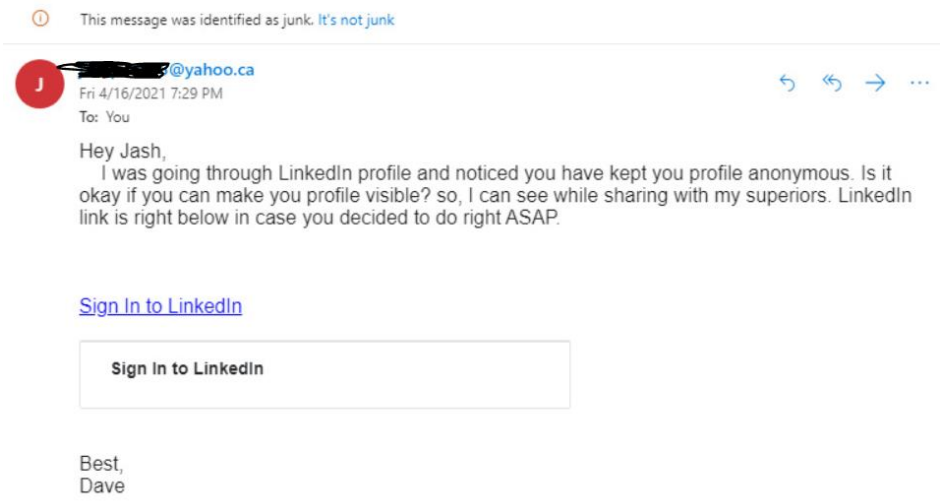

Figure 6: Sent phishing email received

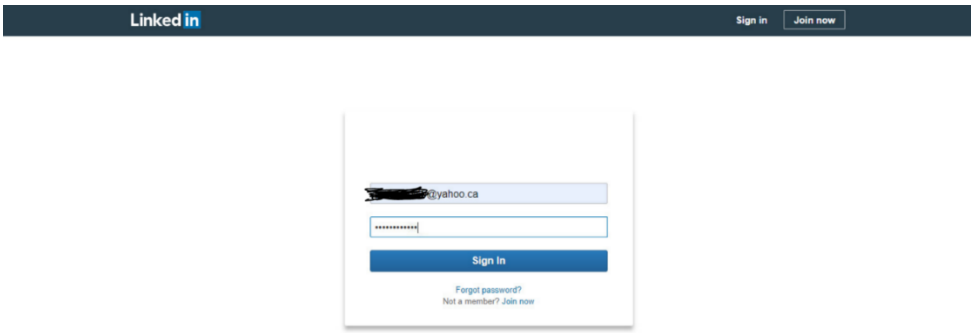

Figure 7: User redirected to phishing site

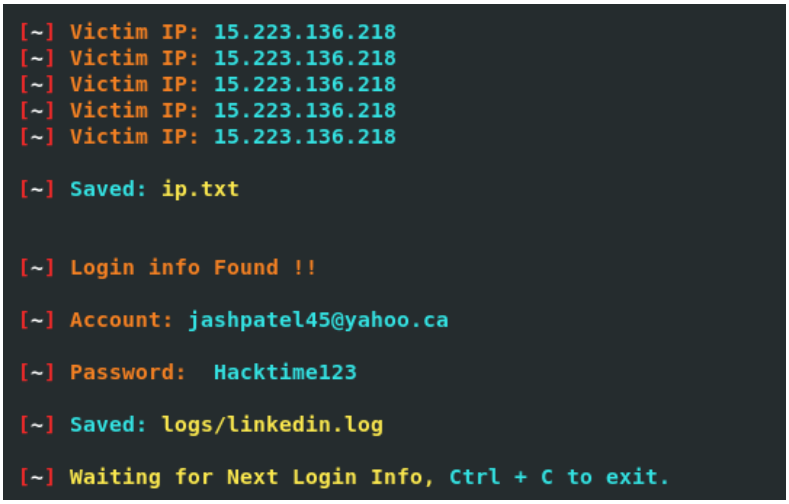

Figure 8: Attacker received the login credentials.

B. Browser Extension

For Part B I created a browser extension that would protect the users from visiting phishing websites and/or exposing their sensitive information to the attacker. The extension follows the framework in figure 3. The extension is written in JavaScript and it uses Web APIs and webpack for node.js to the goals of the phishing detection and prevention extension.

Tools Used for extension:

- Node.js + Webpack
- JavaScript, JSON
- Browsers: Firefox, Chrome and MS Edge
- Publicly available phishing datasets

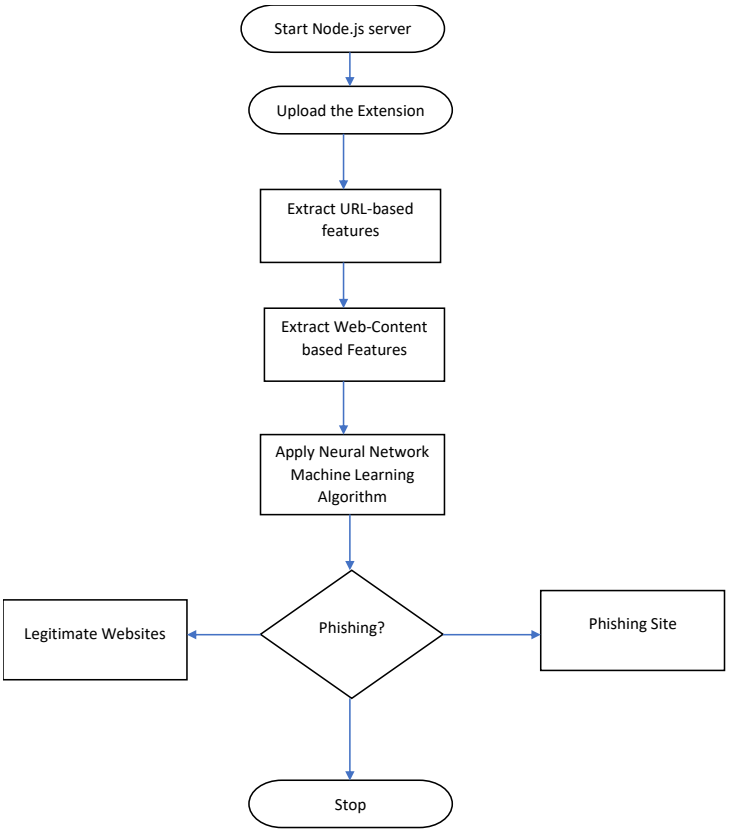

Figure 9: Anti-Phishing Framework

i. Manifest.json File

The first step to creating an anti-phishing extension is creating manifest. json. For any extension to work manifest.json file is mandatory. The manifest file provides essential information about the extension. In other words, it is configuration a file.

manifest.json - Notepad

File Edit Format View Help

```
{
  "name": "Anti-Phisher-Tool",
  "version": "1.0",
  "description": "Secure Web Browsing",
  "manifest_version": 2,
  "browser_action": {
    "default_title": "Anti-Phisher Tool",
    "default_popup": "popup.html"
  },
   "content_scripts": [
  {
    "matches": ["https://developer.mozilla.org/*"],
    "js": ["content.bundle.js"]

  }
],
  "background": {
      "page": "background.html"
  },
   "permissions": ["tabs", "webRequest","webRequestBlocking","activeTab","<all_urls>","file:///*"],

  "web_accessible_resources": ["warning.html"],
  "content_security_policy": "script-src 'self' 'unsafe-eval'; object-src 'self'"


}
```

Figure 10: Manifest.json file

The manifest file includes three files, popup, background and warning file. The main part of the code lies in the background script and warning.html. The background script gets executed as soon as the extension is uploaded which is being used to extract URL and web-content features. While background.html is just a normal warning page that will block phishing websites and popup.html is the front-end of our app. In the picture above, you may notice 'permissions' line is used for giving access to permissions required by the Web APIs that my code uses, for example, XMLHttpRequest().

ii. Web APIs

```
function getContentData(){
  browser.tabs.onActivated.addListener(function (tab) {
    browser.tabs.get(tab.tabId, tabInfo => {
      var url = tabInfo.url;
      //console.log(url);
      const xhr= new XMLHttpRequest();

        getHttpUrlRequest(xhr,url);



    });
  });
}
getContentData();
```

Figure 11: Using http request to extract URL information

The function above is coded to extract URL information from the server upon the select tab. The code not just uses XMLHttpRequest() Web API but, also uses tabs.onActivated () and tabs.get() Web APIs. This is where the 'tabs' and 'request' permissions are needed (refer to manifest file). Although, I had a hard time getting it to work because I was not aware 'request' permission was required at first. In the end, everything worked out well. Therefore, with this function, I was able to get the current browsing URL for domain analysis.

```
function getHttpUrlRequest(xhr,url){
  xhr.open("GET", url,true);

      xhr.onload = () => {
        if (xhr.readyState === 4 && xhr.status == 200) {

          const httpdata= (xhr.response);

          console.log(httpdata);
          extractWebDataFields(httpdata);
```

Figure 12: Getting response data in html format.

My next step was getting the web-content data. I used XMLHttpRequest.open () and XMLHttpRequest.onload () to load web-content data after the request is completed. According to the

image above when the XMLHttpRequest() successfully extracted the web content and the data transferred to the client, it displays the response received in the console and is used for further extracting web-content features.

iii. Feature Extraction

Now, the program is able extract URL and web-content from the active website. The following steps covers the creating rule sets for domain analysis and web-content analysis. Based on mendeley dataset 2021[12], following rules were applied for the phishing sites:

- Length of the URL and hostname
- IP in URL
- Number of dots in domain
- Following number of symbols in the base URL: - @ ? & | =_` % / * : ;$ [space]
- HTTP/s Token, double slash, in URL
- Number of 'www' and 'com' keywords
- Safe anchors: href="#", href="#content", href="#skip", href="JavaScript ::void(0)"
- Number of hyperlinks
- Submit E-Mail
- Domain age

There are total of 28 features that were for training neural networks machine learning model.

iv. Machine Learning: Artificial Neural Networks

```
export const net= new brain.NeuralNetwork();

net.train(test);

output=net.run(features);
console.log(output);
```

Figure 13: Neural networks setup for training and testing

Before implementing the code above, I had to use an open-source webpack plugin to create a localhost node.js server to import data files for testing and must be running before uploading the extension.

There are not many libraries that provide neural networks algorithms in JavaScript but brain.js provided functions that were very easy to set up. Thus, there were not much was required to set up the data. But it uses only .json file to train and test the data. So, I had to convert my .CSV dataset to. json format for brain.js to train and test my data. The figure above shows 'test' is the input that was used to train the data. Using the trained I was able to run the test with extracted features as the output. One of the reasons why I chose this algorithm was because it was stated by [4] to be the most accurate comparison to others classifiers. Artificial Neural Networks (ANNS) is a machine learning model that is

based on a biological neural networks that processes through multi-layered units [10]. Each unit receives input from neighbouring layers and the output resulted in a form weight factor [10].

v. Warning Webpage

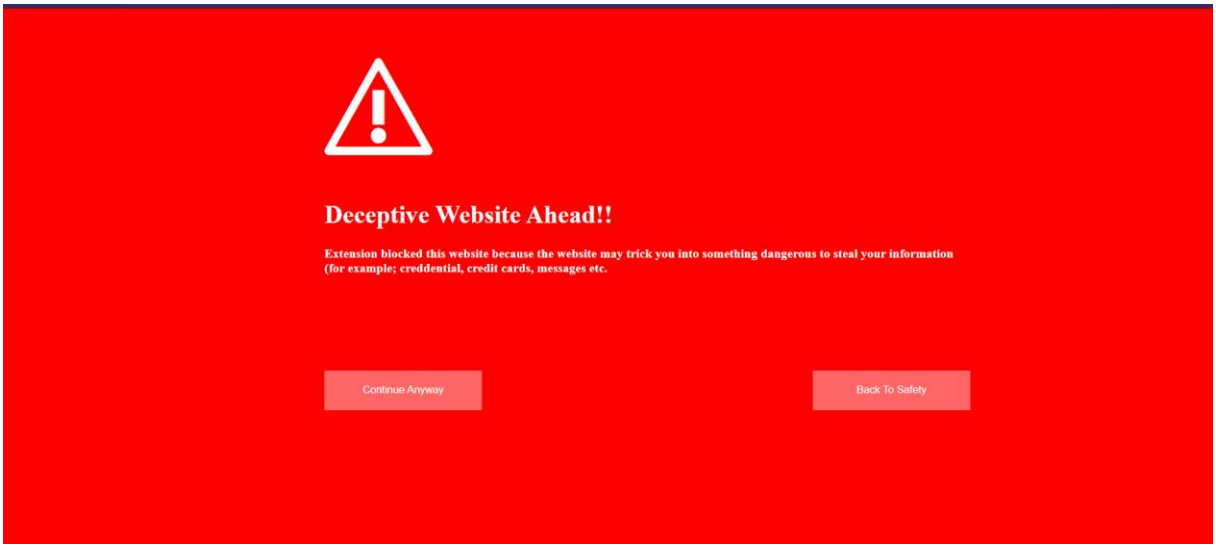

Figure 14: Warning.html page for blocking

The image above is another part of the program. This page is displayed when the user visits phishing websites in a form of blocking. To accomplish this task webrequest.onBeforerequest.addlistener() Web API was required as shown in the following figure.

```
chrome.webRequest.onBeforeRequest.addListener(req,{
        urls: ["<all_urls>"],  types: ['main_frame']
    }, ['blocking']);
```

Figure 15: Add listener before request is made

This Web API an event listener place itself before the header information is available and it gets triggered when a request is about to be made. In other words, it is a good location listen meaning you don't worry if malware gets executed when the user clicks the phishing URL.

vi. Front-end: Popup.html

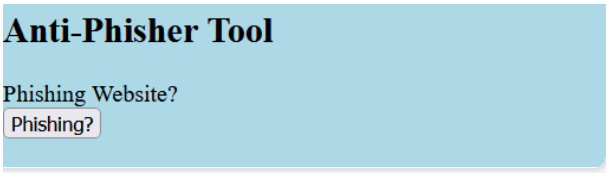

Figure 16: Popup.html

Finally, there is not much for the front-end part of the tool other than adding background colour and buttons. Based on the weight factor received from the brain.js, the result is sending this popup. With 'Phishing?" button you check if the active website is phishing or not. If the weight factor is:

- Greater than 0.70 = Phishing or 1
- Between 0.40-0.69= Suspicious or -1
- Lower than 0.40 = Not Phishing or 0

VIII. Observation and Results

Through the experiment research and proof of concept, many observations were made. Most of the observations were on unsuccessful tasks, error handling, and code-related or environment-related. For example, the browser refused to run request () Web API due to lack of permission or incorrect manifest file configuration. Another failure was sending a phishing link to the victim working in the virtual desktop environment. Based on the proof of concept, the link is supposed to redirect the user to phishing but, unfortunately, Amazon has patched the phishing vulnerability before I can test out my solution. Thus, the following result shows the solution being tested against other phishing websites.

i. Errors, Failures and Permission Issues

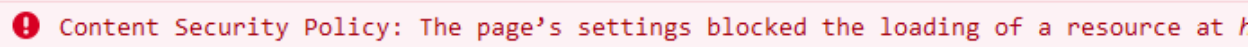

Figure 17: CSP Error

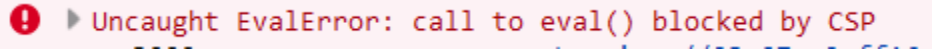

Figure 18: eval() call by webpack plugin

During the development of the project, there were quite a lot of policies that blocked the warning.html file from executing. The figure 17 and 18 are one of them. The Content Security Policy is the policy that the browser is the complaint to follow to mitigate cross-scripting issues [15]. As for eval() is used by webpack running node.js to evaluate the source files and execute them to accomplish its tasks. To fix these issues I had to specify to allow this request in my manifest.json file by adding the following line:

"content_security_policy": "script-src 'self' 'unsafe-eval'; object-src 'self'".

ii. Mendeley Phishing Dataset 2021

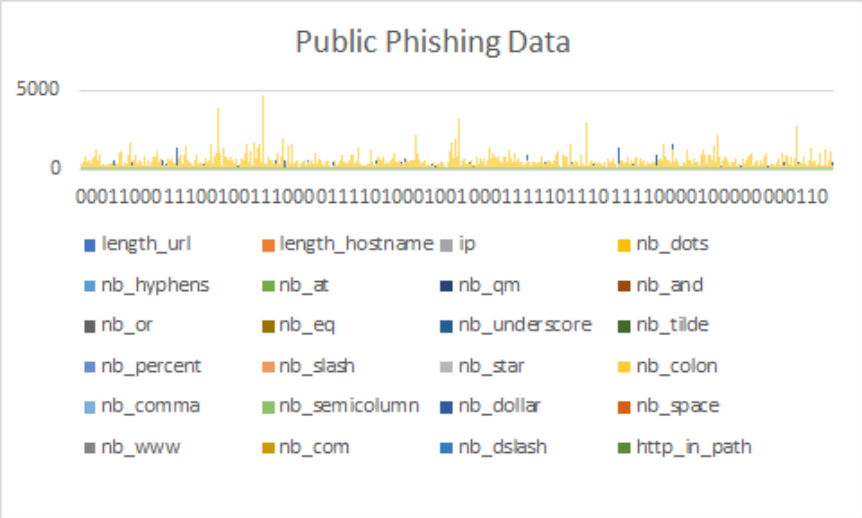

Figure 19: public phishing data

The figure above shows the statistical data from Mendeley phishing dataset 2021. The '0' represent legitimate websites while '1' represent phishing. Based on the result, majority of phishing sites have one thing common that high number of dots in the base URL and the URL length.

iii. Console logging

Logging the result is very important in a business environment. The logs can tell us detailed information about the websites you visited through the machine learning approach. For researching purposes, these logs are not stored in local disk or web storage since they are used for real-time auditing purposes. Tools like AuditLog, Mangoose or Express can be used for capturing events by checking running sessions.

iv. External Phishing Extensions Testing

Tools used: WOT Website Security & Browsing Protection and Zelda Anti-Phishing

The following tables shows 10 phishing and non-phishing websites from the dataset that were tested to compare the results between my anti-phishing tool and other extensions available on the web.

Table 1: Phishing Websites for Testing

Phishing Websites
1. http://www.shadetreetechnology.com/V4/validation/ba4b8bddd7958ecb8772c836c2969531 2. http://appleid.apple.com-app.es/ 3. http://support-appleld.com.secureupdate.duilawyeryork.com/ap/bb14d7ff1fcbf29?cmd=_update&dispatch=bb14d7ff1fcbf29bb&locale=_us

4. http://www.shadetreetechnology.com/V4/validation/ba4b8bddd7958ecb8772c836c2969531 5. http://html.house/l7ceeid6.html 6. http://mno.naomigoff.com/Apollo/bc28a52c2070ca93176244f59e1f19c6 7. http://beta.kenaidanceta.com/postamok/d39a2/source 8. http://www.ktplasmachinery.com/cs/ 9. http://batvrms.net/deliver/D2017HL/u.php 10. http://secureupdate.appleld.com.duilawyeryork.com/ap/88b142e778d73f7
Result
After testing out available extension from chrome webstore the extensions most of websites resulted in as unknown and some were blocked by browser security.

Table 1: Non-Phishing Websites for Testing

Legitimate Websites
1. http://www.whio.com/news/local/thief-breaks-into-save-lot-steals-steaks-dayton/dX1YgaOZ7zmj18bPd4tU6L/ 2. https://www.markspcsolution.com/ 3. https://www.mummyandmini.com/ 4. https://www.youtube.com/watch?v=wo1ySVNZgno 5. http://www.imdb.com/Title?3682448 6. https://www.haciserif.com.tr/ 7. https://www.ablebits.com/office-addins-blog/2014/05/15/excel-insert-hyperlink/ 8. http://techland.time.com/2011/03/15/south-by-southwest-interactive-the-dawn-of-location-based-gaming/ 9. http://www.grafikerler.org/ 10. http://www.wikireality.ru/wiki/Keyboard_Cat
Result
Using WOT Website Security & Browsing Protection extension all of these websites turned out to be safe except website #6 which showed up as unknown. As for Zelda Anti-Phishing, the results were all unknown. Based on my observation I can assume is the Zelda Anti-Phishing tool follows a user training approach with whitelisting and blacklisting.

v. WFH Anti-Phisher Tool

After testing with extension from chrome store, I tested with my anti-phishing extension with the sites from table 1 and table 2. The result showed 90% accuracy for detecting phishing website.

Table 3: Accuracy test

Websites	ANNS: Average Weight Factor	Result
Table 1	0.9090	Phishing
Table 2	0.1898	Not Phishing

From the result above I deduced that there seem to be a lot of False-Positive because the average weight factor for legitimate websites is a little too high. Based on the result by [17], the

researcher had approximately 7% for false-positive rate, while my solution has 19%. This result may help employees working from home to protect against phishing websites but, it does not have 100% security. After all, The extension is trained with recent data.

Another observation I made was the warning page will not display if the browser also considers the active webpage as phishing.

IX. Conclusion

In conclusion, my solution may have been accurate but it is not efficient as I thought it would be. Based on the observation I believe there were a lot of false positives. I thought by applying neutral networks I might get an accurate result but it turned out to be false. There is also too much overload on the browser and it uses too much ram.

X. Areas for Future Study

For future study, I would like to implement this in python where I don't have a deal browser policy, permission, network overload and security risk. I would like to make the code that does not have too many bugs that may cause the accuracy of the tool to be degraded. As an improvement, I would implement multi-layered neural networks with the additional feature to protect embedded malicious attachments in Hotmail/Outlook. That way it is efficient and easy to use for E-Mail and browsing security in a business-like desktop environment.

Bibliography

[1] G. Bassett, C. D. Hylender, P. Langlois, A. Pinto, and S. Widup, “2021 data breach investigations report,” May. 12, 2021. [Online]. Available: https://slate.sheridancollege.ca/content/enforced/851817-1215_42290/2021-data-breach-investigations-report.pdf. [Accessed: Aug. 6, 2021].

[2] S. Van Horn, “Kaspersky report: Criminals Targeted remote work in 2020,” *Business Wire*, Dec. 10, 2020. [Online]. Available: https://www.businesswire.com/news/home/20201210005650/en/Kaspersky-Report-Criminals-Targeted-Remote-Work-In-2020. [Accessed: Aug. 6, 2021].

[3] kaspersky.com, “The dangers of phishing,” Jul. 21, 2017. [Online]. Available: https://go.kaspersky.com/rs/802-IJN-240/images/Dangers_Phishing_Avoid_Lure_Cybercrime_ebook.pdf. [Accessed: Aug. 6, 2021].

[4] Vaitkevičius, Paulius & Marcinkevičius, Virginijus. (2020). Comparison of Classification Algorithms for Detection of Phishing Websites. Informatica. 143-160. 10.15388/20-INFOR404.

[5] K. Chiew, K. Yong, and C. Tan, “A survey of phishing attacks: Their types, vectors and technical approaches,” *Shibboleth authentication request*, 2018. [Online]. Available: https://www-sciencedirect-com.library.sheridanc.on.ca/science/article/pii/S0957417418302070?via%3Dihub. [Accessed: Aug. 6, 2021].

[6] S. Maurya, H. Singh Saini, and A. Jain, “Browser Extension based Hybrid Anti-Phishing Framework using Feature Selection,” *https://thesai.org*, Nov. 30, 2019. [Online]. Available: https://thesai.org/Downloads/Volume10No11/Paper_78-Browser_Extension_based_Hybrid_Anti_Phishing_Framework.pdf. [Accessed: Aug. 6, 2021].

[7] D. Warburton, “2020 phishing and fraud report,” *F5 Labs*, Nov. 11, 2020. [Online]. Available: https://www.f5.com/labs/articles/threat-intelligence/2020-phishing-and-fraud-report. [Accessed: Aug. 6, 2021].

[8] F. Song, Y. Lei, S. Chen, L. Fan, and Y. Liu, “Advanced evasion attacks and mitigations on practical ML-based phishing website classifiers,” *Shibboleth authentication request*, Jun. 14, 2021. [Online]. Available: https://onlinelibrary-wiley-com.library.sheridanc.on.ca/doi/10.1002/int.22510. [Accessed: Aug. 6, 2021].

[9] M. Khonji, Y. Iraqi and A. Jones, "Phishing Detection: A Literature Survey," *in IEEE Communications Surveys & Tutorials.*, vol. 15, no. 4, pp. 2091-2121, Fourth Quarter 2013. doi: 10.1109/SURV.2013.032213.00009.

[10] V. Shahrivari, M. Darabi, and M. Izadi, “Phishing Detection Using Machine Learning Techniques,” Sep. 23,2020. [Online]. Available: https://arxiv.org/pdf/2009.11116.pdf. [Accessed: Aug. 6, 2021].

[11] Cybleinc, “Ngrok platform abused by hackers to deliver a new wave of phishing attacks,” *Cyble*, Jun. 21,2021. [Online]. Available: https://blog.cyble.com/2021/02/15/ngrok-platform-abused-by-hackers-to-deliver-a-new-wave-of-phishing-attacks/. [Accessed: Aug. 6, 2021].

[12] Hannousse, Abdelhakim; Yahiouche, Salima (2021), “Web page phishing detection”, Mendeley Data, V3, doi: 10.17632/c2gw7fy2j4.3

[13] MDN contributors, “webRequest.onBeforeRequest,” *Mozilla*, May. 31, 2021. [Online]. Available: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/API/webRequest/onBeforeRequest. [Accessed: Aug. 6, 2021].

[14] Tania Rascia. 2020. webpack-boilerplate. https://github.com/taniarascia/webpack-boilerplate. (2020)

[15] Chrome Developers, “Content Security Policy,” *Chrome Developers,*May. 14, 2012.[Online]. Available: https://developer.chrome.com/docs/apps/contentSecurityPolicy. [Accessed Aug. 6, 2021]

[16] MDN contributors, “webRequest.onBeforeRequest,” *Mozilla*, May. 31, 2021. [Online]. Available: https://developer.mozilla.org/en-US/docs/Mozilla/Add ons/WebExtensions/manifest.json/content_security_policy. [Accessed: Aug. 6, 2021].

[17] Tanmay Doke, Pranav Khismatrao, Vaibhav Jambhale and Nilesh Marathe, “Phishing-Inspector: Detection & Prevention of Phishing Websites," *ITM Web of Conferences*., Les Ulis, France. Jul. 29, 2020. Vol 32. doi:http://dx.doi.org.library.sheridanc.on.ca/10.1051/itmconf/20203203004.

[18] Kang Leng Chiew, Jeffrey Soon-Fatt Choo, San Nah Sze and Kelvin S.C. Yong, “Leverage Website Favicon to Detect Phishing Websites," *Hindawi Limited*., Les Ulis, 2018. Vol 2018. doi: http://dx.doi.org.library.sheridanc.on.ca/10.1155/2018/7251750.

[19] Akash Kumar, “Phishing URL EDA and modelling,” 2020.

[20] MDN contributors, “XMLHttpRequest,” Mozilla, July 28, 2021. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest. [Accessed: Aug. 6, 2021].

[21] MDN contributors, “tabs.onActivated,” Mozilla, May 31, 2021. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest. [Accessed: Aug. 6, 2021].

[22] Chrome Developers, “Content Security Policy,” *Chrome Developers,*May. 14, 2012.[Online]. Available: https://developer.chrome.com/docs/extensions/mv2/getstarted. [Accessed Aug. 6, 2021]

Appendices

Phishing Prediction

```
if(output >='0.70' && output <= '1.0'){
    console.log("phishing");
    //return 'PHISHING';


    browser.runtime.onMessage.addListener( function(message,sender,sendResponse){

        if(message == 'status'){
            sendResponse("Phishing");
        }
    });
    return requestStatus=1;


}
else if(output < '0.69' && output >= '0.40'){
    console.log("Possibily phishing");


  browser.runtime.onMessage.addListener( function(message,sender,sendResponse){

        if(message == 'status'){
            sendResponse("UNKNOWN/SUSPICIOUS");
        }
    });


}
else{
    console.log("Not a phishing website");
     return 'SAFE';
      browser.runtime.onMessage.addListener( function(message,sender,sendResponse){

        if(message == 'status'){
            sendResponse("SAFE");
        }
    });

}
```

Sending message between warning.html and background.html

```
const browser = window.chrome || window.msBrowser || window.browser;

function redirectUrl(){
    document.getElementById("ignoreRisk").addEventListener("click", function() {


    chrome.runtime.sendMessage('URL sending',function (response) {

    console.log('URL sending',response);
    console.log('recieved');


    window.location.replace(response);

        });



});
}

function redirectToSafety(){
    document.getElementById("backtosafety").addEventListener("click", function() {

    window.location.replace("https://www.google.ca");

    });
}
redirectToSafety();
redirectUrl();
```

Webpack Configuration

```
var options = {
  mode: process.env.NODE_ENV||"development",
  entry: {
    popup: path.join(__dirname,"popup.js"),
    background: path.join(__dirname,"background.js"),
    content: path.join(__dirname,"content.js"),
    warning: path.join(__dirname,"warning.js")
  },
  output: {
    path: path.join(__dirname, "build"),
    filename: "[name].bundle.js"
  },
```

```
    new HtmlWebpackPlugin({
      template: path.join(__dirname,"popup.html"),
          filename: "popup.html",
      chunks: ["popup"]
    }),
    new HtmlWebpackPlugin({
      template: path.join(__dirname,"warning.html"),
      filename: "warning.html",
      chunks: ["warning"]
    }),

    new HtmlWebpackPlugin({
      template: path.join(__dirname, "background.html"),
      filename: "background.html",
      chunks: ["background"]
    }),
    new WriteFilePlugin()
  ]
};
```

Rest of the additional work is in the following github:
https://github.com/jashpatel4533/Capstone2021.git